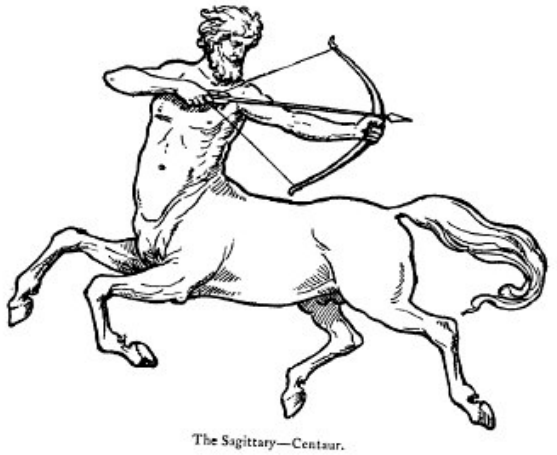
ІНФОРМАТИКА ТА ПРОГРАМУВАННЯ

Тема 18. Множинне наслідування

Множинне наслідування

- Наслідування називають **множинним**, якщо клас наслідує від більш, ніж одного, батьківського класу.
- Приклади множинного наслідування ми бачимо у оточуючому житті.
- Наприклад, смартфон можна назвати спадкоємцем мобільного телефону та операційної системи.
- У міфології відомі такі фантастичні створіння, як кентаври або грифони, які наслідують від різних істот.


The Sagittary—Centaur.

Множинне наслідування.2

- Множинне наслідування не настільки розповсюджене, як одинарне, але є задачі, у яких множинне наслідування є природним та дозволяє зробити текст програм коротшим та більш зрозумілим.

Синтаксис множинного наслідування

- Якщо клас D наслідує від класів B та C, то це записують наступним чином:

```
class D (B, C):
    ...
```

- У цьому випадку клас D успадковує всі поля та методи класів B та C.

Приклад: гра у відгадування слів зі збереженням даних

- Реалізувати гру у відгадування слів, яка полягає у наступному.
- По колу розташовані гравці (відгадувачі), яким презентують слово для відгадування.
- Всі літери цього слова спочатку закриті (замінені зірочками, ‘*’).
- Гравці вступають у гру по порядку.
- Кожен гравець може назвати літеру або слово (повний опис гри міститься у темі «Рекурсивні структури даних»).

- Забезпечити збереження даних про відгадування слів: гравців та їх результатів.

Реалізація гри у відгадування слів зі збереженням даних

- Для реалізації цього прикладу використаємо множинне наслідування від кільцевого списку та класу збереження/відновлення даних.
- Опишемо клас LoadSave, який зберігає у файлі та відновлює з файлу дані атрибутів деякого класу.
- Цей клас містить поля
  - filename - ім'я файлу для збереження даних;
  - __attribute_names - список імен атрибутів, які будуть збережені;

а також методи save() та load() для збереження та відновлення даних.

Реалізація гри у відгадування слів зі збереженням даних.2

```
class LoadSave:
    '''Клас, який зберігає дані з атрибутів класу-нащадка    '''
    def __init__(self, filename, *attribute_names):
        '''Конструктор збирає у список імена атрибутів.
        Проводить перейменування атрибутів, якщо потрібно
(починаються з __).
        '''
        self.filename = filename        #ім'я файлу для збереження даних
        self.__attribute_names = []
                        #список імен атрибутів, які будуть збережені
        for name in attribute_names:
            if name.startswith("__"):
                name = "_" + self .__class__.__name__ + name
            self.__attribute_names.append(name)
```

Реалізація гри у відгадування слів зі збереженням даних.3

```
def save(self):
    '''Зберігає дані у файлі.

    '''
    with open(self.filename, "wb") as fh:
        data = []                          #список значень атрибутів
        for name in self.__attribute_names:
            data.append(getattr(self, name))
#додати значення атрибуту до списку
        pickle.dump(data, fh, pickle.HIGHEST_PROTOCOL)
#зберегти список
```

Реалізація гри у відгадування слів зі збереженням даних.4

```
def load(self):
    '''Читає дані з файлу.

    '''
    with open(self.filename, "rb") as fh:
        data = pickle.load(fh) #прочитати збережений список
        for name, value in zip(self.__attribute_names, data):
            setattr(self, name, value) #змінити значення
атрибуту
```

- Клас LoadSave для роботи з файлами використовує pickle та менеджер контексту with.

Менеджер контексту with

- Менеджер контексту є аналогом блоку try – except – finally, який ми розглядали у темі «Обробка помилок та виключних ситуацій».
- Синтаксис менеджера контексту виглядає так:

```
with e as a:
    P
```

  - де *e* – вираз, *a* – змінна, *P* – інструкція.
- Вираз *e* повинен повертати об'єкт, який підтримує протокол менеджменту контексту.
- Цей об'єкт присвоюється змінній *a*, після чого Python виконує інструкцію *P*.
- Якщо інструкція *P* міститься всередині блоку with, це гарантує звільнення спільних ресурсів після завершення блоку, навіть у випадку виключення.
- Зокрема, менеджери контексту використовують у обробці файлів (файлові об'єкти підтримують протокол менеджменту контексту).
- Файл, який відкрито у заголовку with, буде обов'язково закритий після його завершення.

Реалізація гри у відгадування слів зі збереженням даних. Продовження

- Опишемо також клас FileRlist, який наслідує від класів Rlist та LoadSave.
- У цьому класі перевизначений тільки конструктор __init__

```
class FileRlist(Rlist, LoadSave):
    '''Клас, який успадковує від Rlist, LoadSave
    '''
    def __init__(self, filename):
        Rlist.__init__(self)
        LoadSave.__init__(self, filename, "_lst", "_cur")
        #будуть збережені значення атрибутів "_lst",
"_cur"
```

Реалізація гри у відгадування слів зі збереженням даних. Продовження.2

- Основний модуль імпортує функції з модуля wordguess, реалізованого у темі «Рекурсивні структури даних», а також розглянуті вище класи з модуля loadsaverlist.

Доступ до полів та методів батьківських класів при множинному наслідуванні

- Повернемось до прикладу класу

```
class D (B, C):
    ...
```

- Розглянемо, як забезпечується доступ до полів та методів класу D та його предків. Наприклад, для

```
x = D()
```

- Цей доступ може бути потрібним у реалізації самого класу або для об’єктів класу.
- У другому випадку, з точки зору синтаксису, нічого не змінюється у порівнянні з одиничним наслідуванням: ми повинні вказати метод, який є у класі (власний або успадкований).
- Наприклад,

```
x.meth()
```

- якщо сам клас або його предки містять метод meth().

Доступ до полів та методів батьківських класів при множинному наслідуванні.2

- У першому випадку (у реалізації самого класу) можна явно вказати ім’я класу, як ми вже це робили раніше, або використати функцію super.
- Наприклад, якщо у конструкторі класу D (__init__) нам треба звернутись до конструктора класу B, ми можемо просто вказати B.__init__( ).
- Інший спосіб полягає у використанні super: super().__init__( ).
- Взагалі, синтаксис super має такий вигляд:

```
super(cls, obj).meth(...)
```

  - де cls – ім’я класу, obj – об’єкт, для якого викликається метод meth.
- Якщо клас та об’єкт не вказано, мається на увазі поточний клас.
- super викликає метод, який зазвичай належить батьківському класу.

Співставлення методів об’єктам

- Взагалі, який саме метод, якого класу викликається для того чи іншого об’єкту, залежить від співставлення методів об’єктам.
- Це співставлення здійснюється під час виконання програми у порядку співставлення методів (Method Resolution Order або MRO).
- MRO визначає порядок, у якому Python шукає метод, що повинен бути застосований до об’єкту під час виклику цього методу.
- Якщо цей метод є у класі, до якого відноситься об’єкт, то все просто: саме цей метод застосовується.
- Але якщо такого методу у класі немає, Python шукає цей метод у порядку, визначеному MRO.
- Для одинарного наслідування порядок пошуку такого метода – це пошук у батьківських класах даного класу вгору по ієрархії класів.
- В той же час, для множинного наслідування все не так просто.
- У загальному випадку, для множинного наслідування алгоритм MRO не є простим та очевидним, але, у більшості випадків, для множинного наслідування пошук метода здійснюється у всіх класах, від яких безпосередньо наслідує даний, зліва направо в порядку вказання цих класів у списку класів-предків даного класу.

Співставлення методів об’єктам.2

- Так, якщо клас D наслідує від B та C

```
class D (B, C):
    ...
```

- то Python після пошуку у класі D спочатку буде шукати потрібний методу у класі B, а потім, - у класі C, і вже після цього – у класах-предках B та C.
- MRO застосовується і для явного вказання імен методів, і для використання super, оскільки super викликає саме наступний метод у порядку MRO.
- Для того, щоб дізнатись порядок співставлення методів для деякого класу, можна використати поле __mro__ для цього класу.
- Значенням цього поля є список класів для співставлення методів у порядку MRO. Наприклад,

```
D.__mro__
```

Обмеження використання множинного наслідування

- Залежність порядку співставлення методів від порядку вказання класів у списку предків а також потенційні проблеми зі співставленням методів у класах-нащадках (які можуть змінити порядок MRO) обмежують використання множинного наслідування та функції super.
- Фахівці рекомендують радше відмовитись від множинного наслідування (у меншій степні) та від функції super (у більшій степені).
- У будь-якому випадку, слід обережно відноситись до використання super та від множинного наслідування та використовувати їх у випадках, коли це використання є виправданим.

Множинне наслідування та класи-«домішки»

- Однією з цікавих сфер застосування множинного наслідування є використання класів-«домішків» (mixins).
- **Домішки** – це класи, які додають до будь-якого іншого класу визначену функціональність через механізм множинного наслідування.
- Наприклад, опис

```
class D (MixinCls, C):
    ...
```

- з використанням домішку MixinCls надає класу D, як нащадку C, додаткову функціональність.
- Оскільки ця функціональність може бути додана до будь-якого класу, вона зазвичай є доволі узагальненою.
- Особливість домішків у тому, що вони, як правило, не мають власних полів та не містять власний метод _init_.

Приклад: перевірка, чи є граф деревом, та відслідковування дій над графом

- Перевірити, чи є граф деревом.
- Здійснити відслідковування виконання дій над графом.

- Деревом називають зв’язний граф з одним джерелом та напівстепінню входу всіх вершин не більше 1.

- Для реалізації цього завдання використаємо клас GraphIt – граф з ітератором – описаний у темі «Ітератори та генератори».
- У цьому класі також є методи, що повертають напівстепінь входу та виходу вершини графу: hdegin, hdegout.

Реалізація перевірки, чи є граф деревом, з використанням домішків

- Окрім класу GraphIt, використаємо також клас-домішок LoggedMappingMixin, що здійснює виведення факту використання операцій читання, зміни та видалення елементу деякого типу даних.
- Цей клас містить перевизначення спеціальних методів __getitem__, __setitem__ та __delitem__, які відповідають за виконання вищевказаних функцій.

Реалізація перевірки, чи є граф деревом, з використанням домішків.2

```
class LoggedMappingMixin:
    '''Додати виведення операцій get/set/delete для
налагодження.
    '''
    def __getitem__(self, key):
        print('Getting ' + str(key))
        return super().__getitem__(key)

    def __setitem__(self, key, value):
        print('Setting {} = {!r}'.format(key, value))
        return super().__setitem__(key, value)

    def __delitem__(self, key):
        print('Deleting ' + str(key))
        return super().__delitem__(key)
```

Реалізація перевірки, чи є граф деревом, з використанням домішків.3

- У кінці кожного з перевизначених методів йде виклик наступного у порядку MRO методу за допомогою super.
- Наприклад, super().__delitem__(key).
- Опишемо також реалізацію класа-нащадка LoggedGraph, який успадковує від GraphIt та LoggedMappingMixin.
- Ця реалізація є тривіальною що характерно для використання домішків.

```
class LoggedGraph(LoggedMappingMixin, GraphIt):
    '''Клас, що успадковує від GraphIt та домішку
LoggedMappingMixin.    '''
    pass
```

Реалізація перевірки, чи є граф деревом, з використанням домішків.4

- Для того, щоб при виклику, скажімо, __getitem__ функція super з класу LoggedMappingMixin правильно викликала відповідний метод з класу GraphIt, необхідно використовувати саме такий порядок класів у списку класів предків LoggedGraph:
  - LoggedMappingMixin, GraphIt.
- Модуль, який містить описи класів LoggedGraph та LoggedMappingMixin, також використовує функції з побудови графу із файлу fileinputgraph та перевірки, чи є граф деревом istree.
- Ці функції були описані у темах «Рекурсивні структури даних» та «Ітератори та генератори».

Резюме

- Ми розглянули:
  1. Множинне наслідування. Синтаксис множинного наслідування
  2. Менеджер контексту with
  3. Функція super
  4. Доступ до методів класів –предків.
  5. Порядок співставлення методів (MRO)
  6. Обмеження множинного наслідування та super
  7. Класи-домішки (mixins)

Де прочитати

1. Марк Лутц, Изучаем Python, 4-е издание, 2010, Символ-Плюс
2. Python 3.4.3 documentation
3. Марк Саммерфилд, Программирование на Python 3. Подробное руководство. - Символ-Плюс, 2009.
4. Bruno R. Preiss, Data Structures and Algorithms with Object-Oriented Design Patterns in Python, 2003, http://www.brpreiss.com/books/opus7/
5. Tarek Ziadé. Expert Python Programming. - Packt Publishing, 2008.
6. David Beazley and Brian K. Jones, Python Cookbook. - O'Reilly Media, 2013.
7. http://habrahabr.ru/post/62203/